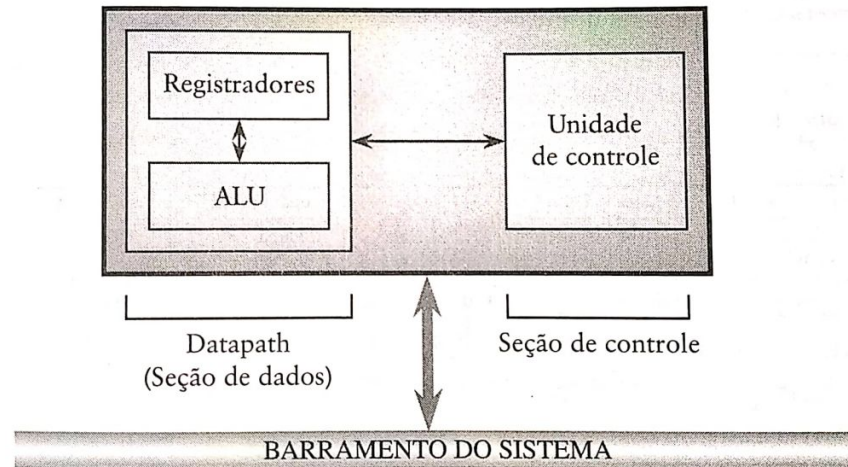
O MICROPROCESSADOR GENÉRICO

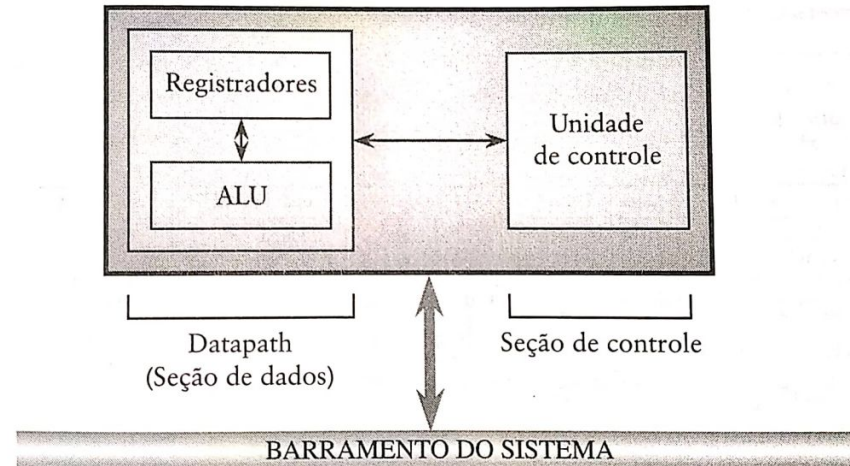
Professor: Alan Sovano, M.Sc.
Disciplina: Microprocessadores (2024.4)

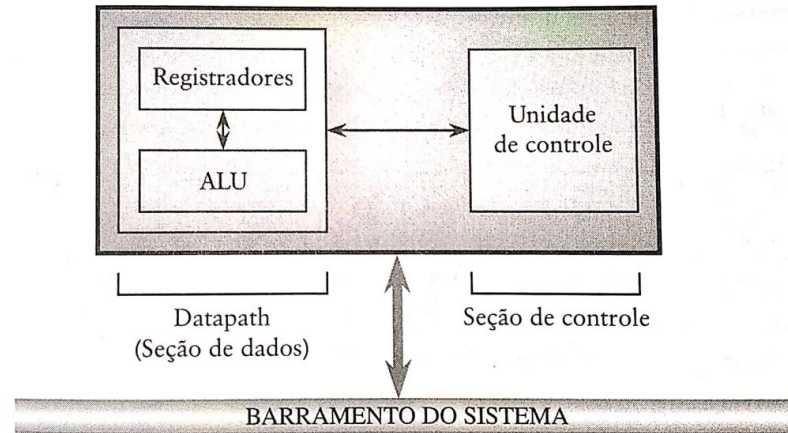
TÓPICOS

- Estrutura simplificada de um microprocessador
- O ciclo de instrução de um microprocessador
- Microarquitetura interna de um microprocessador genérico
- Tipos de dados
- *Pipelining*
- Processadores com múltiplos *cores* e múltiplos *threads*

ESTRUTURA SIMPLIFICADA DE UM MICROPROCESSADOR

- Um **microprocessador**, ou simplesmente **processador**, pode ser descrito, de forma simplificada, a partir da figura abaixo:

ESTRUTURA SIMPLIFICADA DE UM MICROPROCESSADOR

- Esse sistema possui duas áreas distintas: a primeira é a área de **processamento de dados**, a qual é composta por um conjunto de **registradores** e uma **unidade lógica e aritmética** (ou ULA/UAL/ALU). Ela realiza todas as operações lógicas e aritméticas.

ESTRUTURA SIMPLIFICADA DE UM MICROPROCESSADOR

- A segunda área é a da **unidade de controle** (UC), responsável por: sincronizar todo o sistema computacional; enviar comandos para memória e/ou dispositivos de E/S; decodificar instruções que serão interpretadas por microcódigo ou executadas diretamente via hardware.

ESTRUTURA SIMPLIFICADA DE UM MICROPROCESSADOR

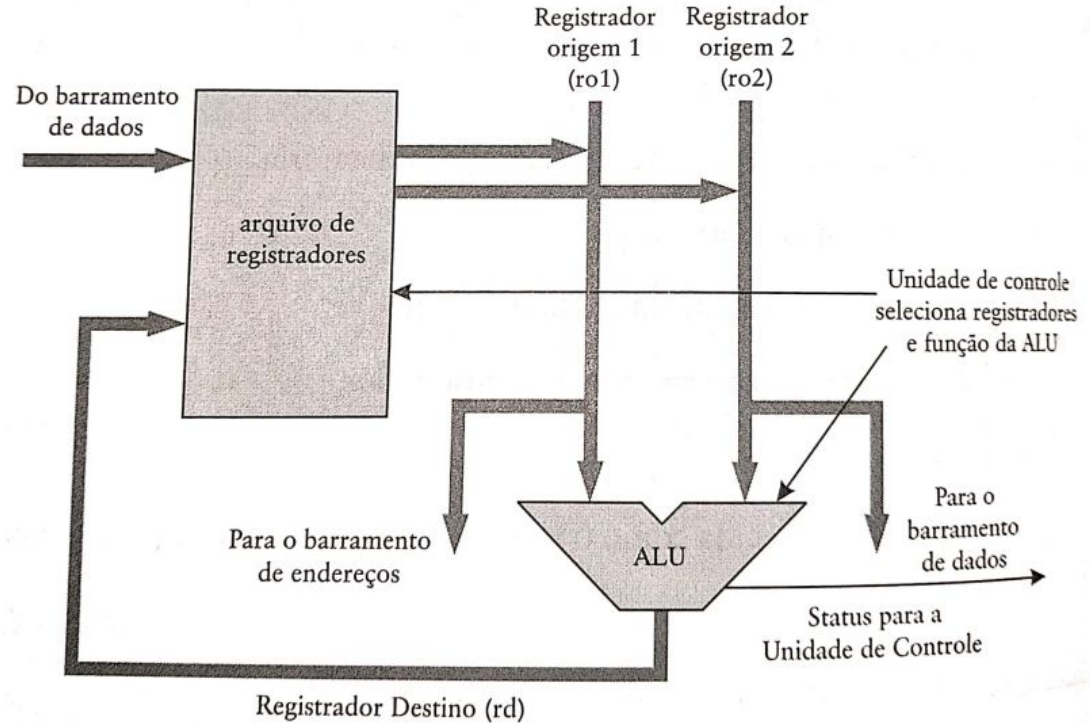
- A figura ao lado ilustra o **caminho dos dados** ou o ***datapath*** de uma UCP (Parte da UCP composta pela ULA junto com suas entradas e saídas);

- O conjunto de registradores da UCP pode ser chamado de **arquivo de registradores** ou de ***register-file***.

OBS: Lembre-se: os registradores são circuitos sequenciais formados a partir de flip-flops!

OBS2: O arquivo de registradores é, geralmente, associado somente aos registradores da parte de processamento. Entretanto, algumas literaturas e fabricantes atribuem o título "arquivo de registradores" a **todo** o conjunto de registradores do sistema (incluindo os de controle). Estamos considerando, aqui, o segundo caso.

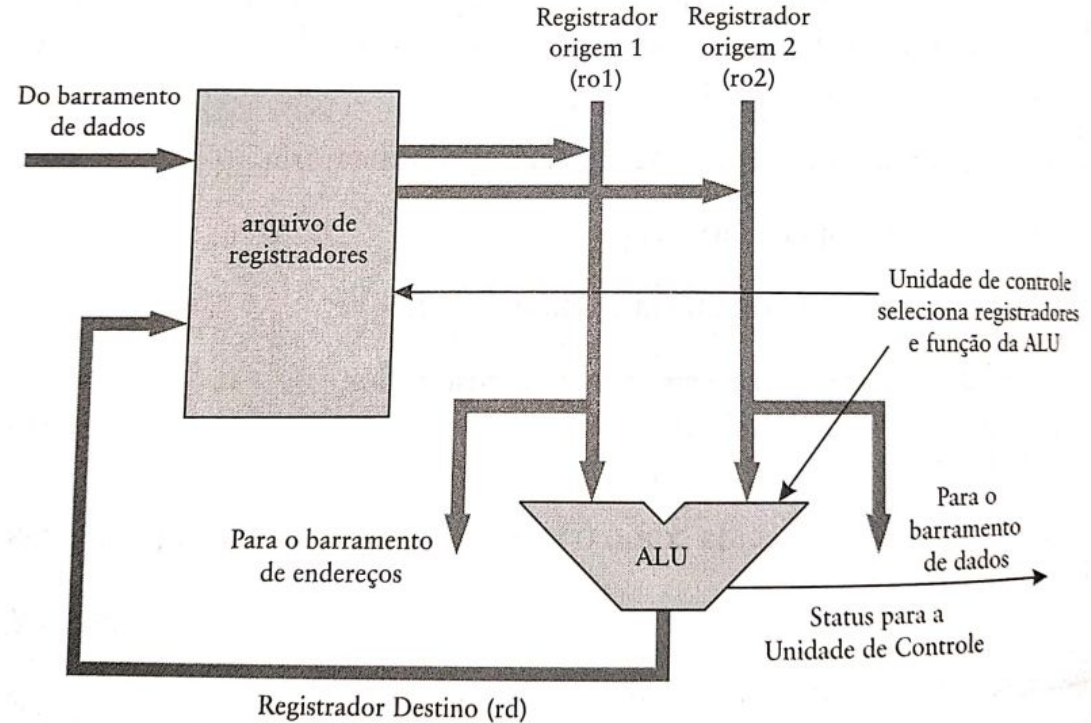
Registrador origem 1 (ro1)
Registrador origem 2 (ro2)
Do barramento de dados
arquivo de registradores
Unidade de controle seleciona registradores e função da ALU
Para o barramento de endereços
ALU
Para o barramento de dados
Status para a Unidade de Controle
Registrador Destino (rd)

Prof. Alan Sovano

ESTRUTURA SIMPLIFICADA DE UM MICROPROCESSADOR

- Na UCP, os dados (vindos através do barramento de dados) são armazenados, temporariamente, em registradores, os quais enviam a informação para a ULA. O resultado do processamento é enviado para um outro registrador, onde fica armazenado;

- A partir daí, o dado pode ser utilizado em um novo cálculo ou pode ser salvo na memória principal. A ULA pode ter também operado números cujo resultado seja um endereço, o qual deve ser enviado pelo barramento de endereços.

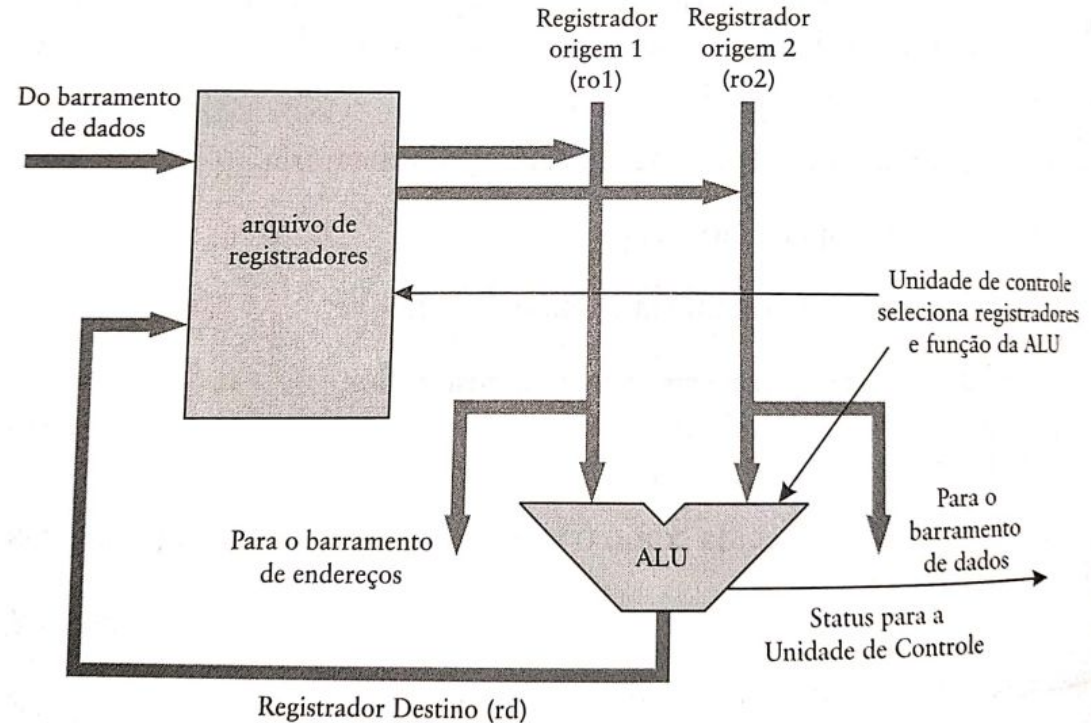
Registrador origem 1 (ro1)
Registrador origem 2 (ro2)
Do barramento de dados
arquivo de registradores
Unidade de controle seleciona registradores e função da ALU
Para o barramento de endereços
ALU
Para o barramento de dados
Status para a Unidade de Controle
Registrador Destino (rd)

Prof. Alan Sovano

ESTRUTURA SIMPLIFICADA DE UM MICROPROCESSADOR

- A unidade de controle envia para a ULA qual operação ela deve realizar (uma soma, subtração, operação AND, operação NOR, entre outras possíveis);

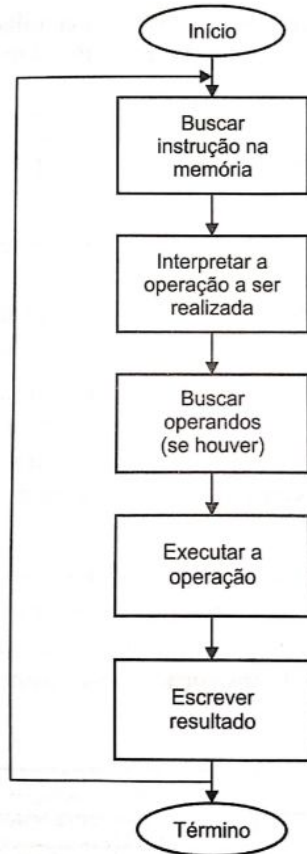
- Ela também define quais os registradores que devem ser acessados para determinada operação, dita o sincronismo do sistema e envia sinais de controle para os outros componentes, de forma a fazer com que cada um funcione na hora certa e do jeito certo.

Registrador origem 1 (ro1)
Registrador origem 2 (ro2)
Do barramento de dados
arquivo de registradores
Unidade de controle seleciona registradores e função da ALU
Para o barramento de endereços
ALU
Para o barramento de dados
Status para a Unidade de Controle
Registrador Destino (rd)

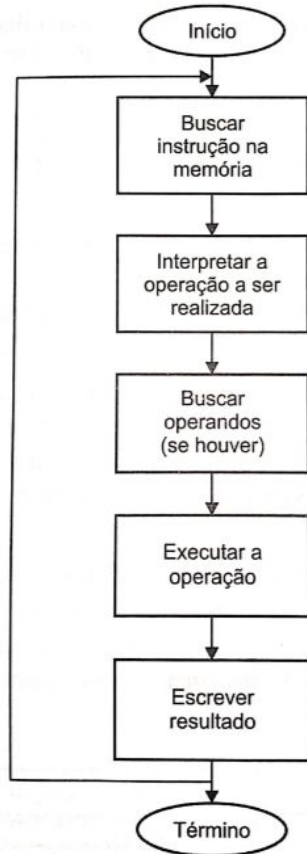
Prof. Alan Sovano

O CICLO DE INSTRUÇÃO DE UM MICROPROCESSADOR

- Como um microprocessador executa uma instrução? Quais são os "passos" que ele segue ao receber um comando e gerar uma saída?

- A sequência de ações que um microcontrolador realiza para executar uma instrução é o chamado **ciclo de instrução**, conhecido também como **ciclo de busca-decodificação-execução (*fetch-decode-execute cycle*)** ou, simplesmente, **ciclo de busca-execução** (***fetch-execute cycle***).

O CICLO DE INSTRUÇÃO DE UM MICROPROCESSADOR

- A figura ao lado ilustra, basicamente, como ocorre o ciclo de instrução:
- Quando solicitamos que o processador execute um programa, ele vai buscar na memória qual a primeira instrução que deve ser executada;
- Em seguida, ele irá decodificar a instrução e gerar os sinais de controle apropriados para que a instrução seja executada. Esses sinais de controle podem ser **gerados** de **forma direta**, utilizando um **circuito combinacional**, ou por meio de uma **interpretação** das instruções, a qual ocorre com o auxílio de um **microcódigo**;
- No próximo momento, ele vai buscar (caso existam) os **operandos** da instrução, ou seja, os dados que serão utilizados na realização das operações solicitadas.

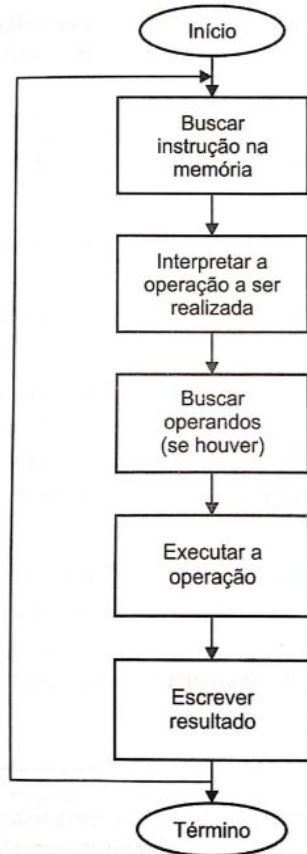
Prof. Alan Sovano

O CICLO DE INSTRUÇÃO DE UM MICROPROCESSADOR

- Após saber a operação que ele deve realizar e quem são os dados que devem ser operados, o processador realiza a operação em si;

- Por fim, após a execução, o processador retorno o resultado final da operação;

- Após o resultado final ser gerado, o processador começa todo o processo novamente.

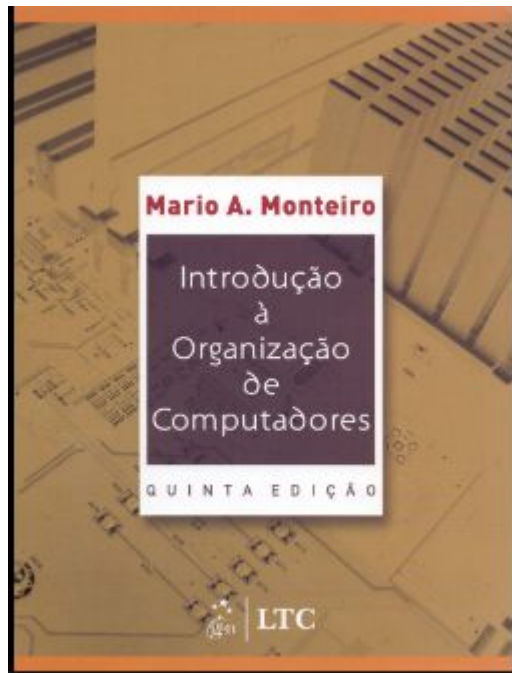
Prof. Alan Sovano

O CICLO DE INSTRUÇÃO DE UM MICROPROCESSADOR

- No fluxograma, a indicação de "início" representa, por exemplo, o momento em que ligamos o computador, enquanto o "término" representa o momento em que desligamos;

- O processador, portanto, segue esse passo a passo várias vezes durante o período em que estiver processando informações, parando somente ao ser desligado, reiniciado ou receber um explicitamente uma instrução de parada;

- A ordem de execução de várias instruções é definida pela organização do programa que foi repassado ao processador.

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Não há, entre os livros da área de arquitetura de computadores/microprocessadores, um consenso entre o que seria uma arquitetura básica, genérica e ideal – cada livro costuma criar uma arquitetura didática e discutir vários conceitos a partir dela;

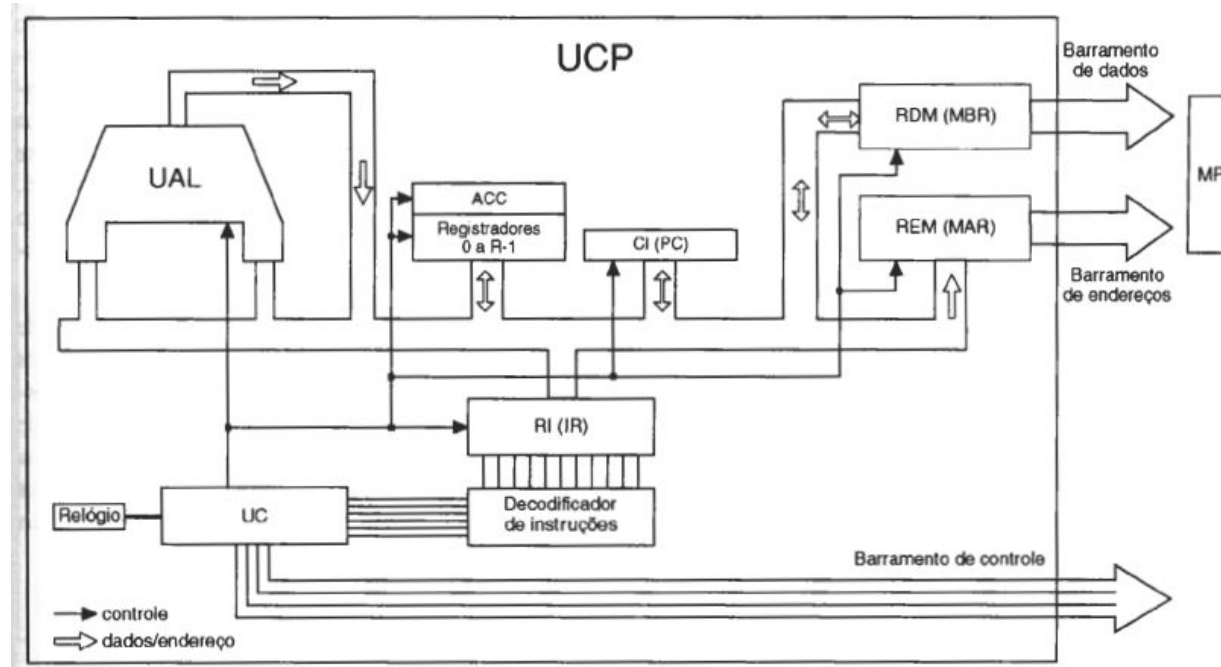
- Em nosso caso, iremos utilizar, nesse momento, o livro "Introdução à Organização de Computadores" (5ª edição), do Prof. Monteiro, o qual mostra um sistema bem genérico, simples e poderemos utilizar como base;

- Em aulas futuras, iremos nos aprofundar em uma dessas arquiteturas didáticas, a qual chamaremos de SAP (*Simple-As-Possible Computer*).

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

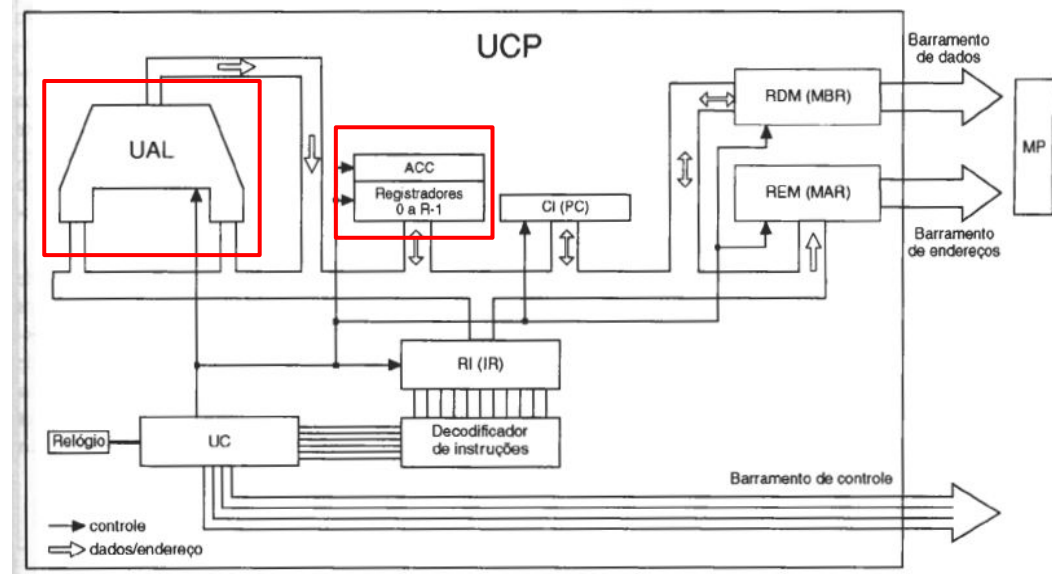
- Temos, na figura abaixo, os componentes principais de um microprocessador, dentre os quais podemos destacar a ULA/UAL e a UC. Além disso, é possível visualizar vários registradores, o decodificador de instruções e o relógio do sistema.

UCP
UAL
ACC
Registradores 0 a R-1
CI (PC)
RDM (MBR)
REM (MAR)
Barramento de dados
MP
Barramento de endereços
RI (IR)
Relógio
UC
Decodificador de instruções
Barramento de controle
controle
dados/endereço

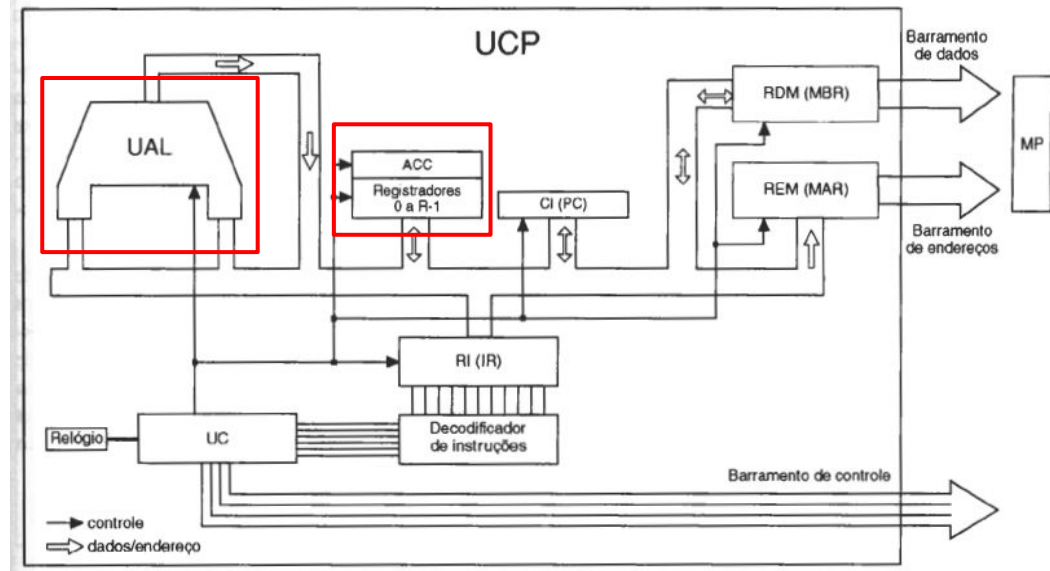
Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Os componentes marcados em vermelho destacam os blocos de processamento (a ULA/UAL e os registradores utilizado para armazenamento de dados);

- Nesse sistema, é mostrado um registrador especial chamado de ACC (acumulador). Um registrador acumulador guarda resultados intermediários que serão utilizados novamente no programa (seu uso ficará mais claro quando falarmos do nível ISA);

- Nesse momento, o que importa é saber que o registrado ACC e os registradores de 0 a R-1 são todos registradores de dados.

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Vale ressaltar que um processador não precisa, necessariamente, ter uma única ULA/UAL, podendo ter várias para realizar múltiplas operações ao mesmo tempo;

- O ATMega328p, por exemplo, possui uma única ULA/UAL em sua UCP, enquanto um Intel Core i7 de 13ª geração possui múltiplas ULAs/UALs.

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Na figura ao lado, temos as entradas e saídas de uma ULA/UAL;
- Os **sinais de controle** determinam qual operação será realizada pela ULA/UAL, enquanto as **entradas de registradores** são as entradas de dados em si;
- O resultado da operação é direcionado à **saída para registradores**. Dependendo do resultado e da operação, valores diferentes são transmitidos à **saída para *flags*** e chegam ao **registrador de controle** (**OBS:** esse registrador não aparece na figura do slide anterior e nem na figura ao lado, mas aparece na figura do próximo slide).

Figura 6.10 Interligação da UAL ao restante da UCP.

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

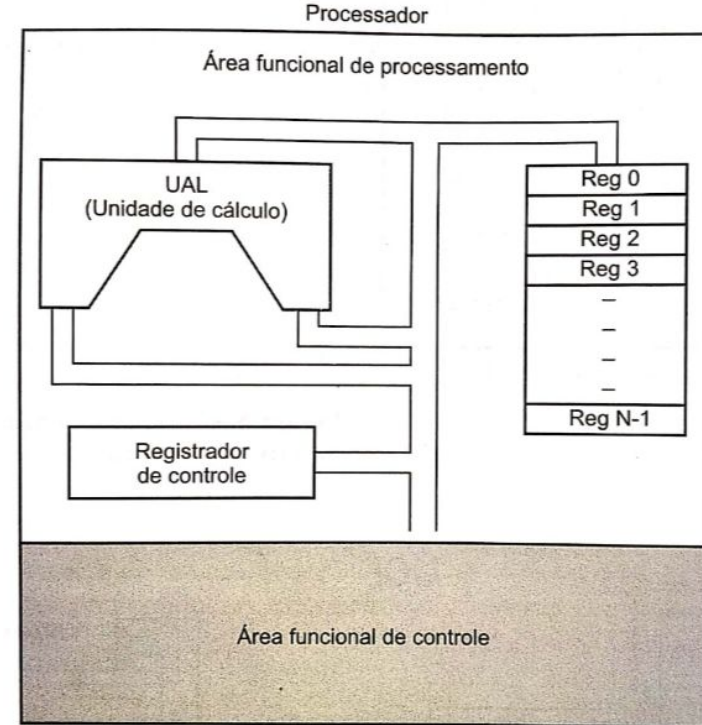
- Uma ***flag*** é uma "sinalização" para o sistema de que algum processo específico ocorreu;

- A ULA/UAL pode usar um *flag* para indicar que ocorreu, por exemplo um *carry* ou um *overflow* em uma operação aritmética específica;

- A ULA/UAL envia uma *flag* para o sistema como um conjunto de bits, os quais são salvos em um registrador específico para esse fim;

- As *flags* indicam ações específicas que a unidade de controle deve tomar para continuar com o processamento da informação.

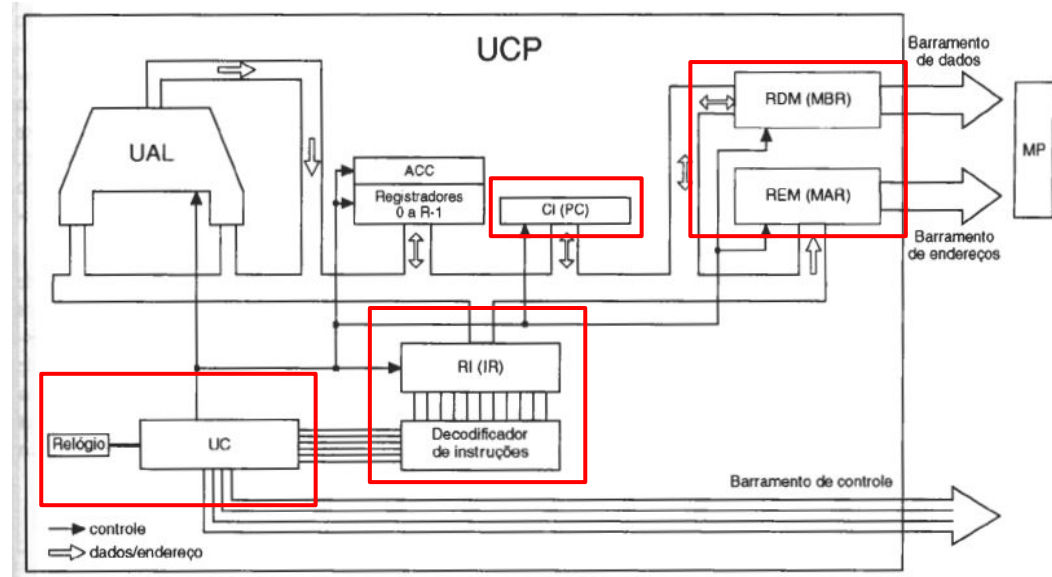
OBS: Não existem somente *flags* geradas pela ULA/ALU. Elas também podem ser setadas diretamente através de um programa ou por dispositivos de E/S (**exemplo:** *flag* para indicar uma interrupção do sistema).

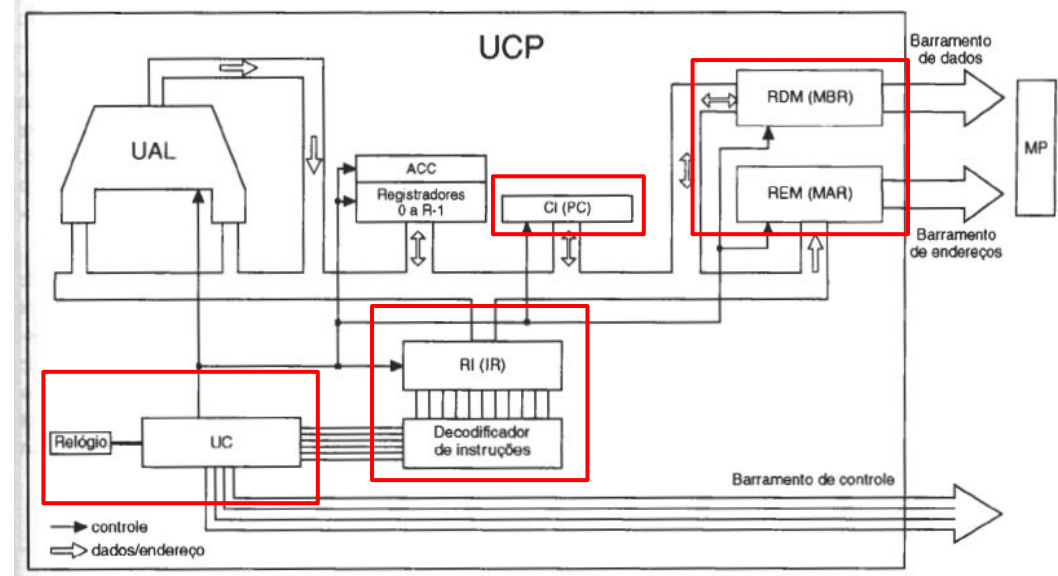
Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Agora, em nossa figura que ilustra a microarquitetura de um processador genérico, temos destacados os blocos que participam das tarefas de controle do sistema;

- Entre as tarefas de controle, estão a busca (na memória principal) da instrução que deve ser executada e a decodificação dessa instrução (com o auxílio de um microcódigo ou de um circuito combinacional). Essas etapas constituem o *ciclo de busca da instrução (fetch cycle)*, junto com o *ciclo de decodificação da instrução (decode cycle)*;

- A instrução, ao ser decodificada, se transforma em vários sinais de controle para o resto do sistema, ordenando o que cada componente deve fazer a cada instante de tempo para que a instrução seja realizada com sucesso. Esse é o *ciclo de execução da instrução (execute cycle)*;

- A unidade de controle, portanto, é a responsável principal por ordenar o ciclo de busca-decodificação-execução.

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Na figura ao lado, temos em destaque:
  - O relógio (*clock*) do sistema;
  - A UC (unidade de controle) propriamente dita;
  - O decodificador de instruções;
  - O Registrador de instruções (RI) ou *Instruction Register (IR)*;
  - O Contador de Instruções (CI), *Program Counter (PC)* ou, ainda, *Instruction Pointer* (IP);
  - O Registrador de Dados de Memória (RDM) ou *Memory Buffer Register (MBR)*;
  - O Registrador de Endereços de Memória (REM) ou *Memory Address Register (MAR)*.

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

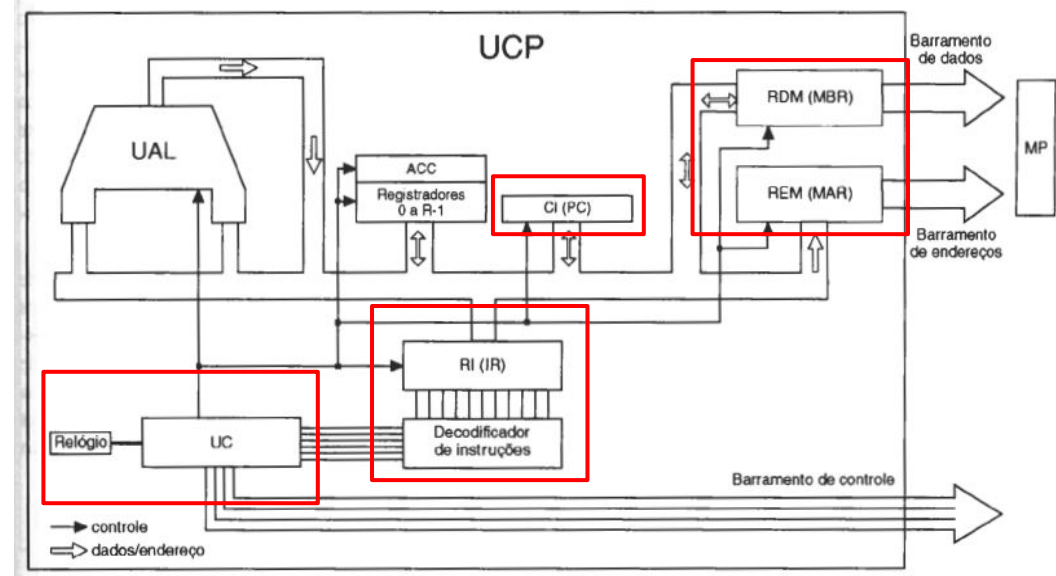
- A UC gera vários sinais de controle para o resto do sistema de forma temporizada, onde o ciclo de instrução é executado em uma ou mais sub etapas, as quais são chamadas de *microeventos* ou *microoperações*;

- A microoperação é a menor operação possível que um processador pode realizar (por exemplo, passar a informação de um registrador para outro ou guardar a informação do barramento de dados em um registrador);

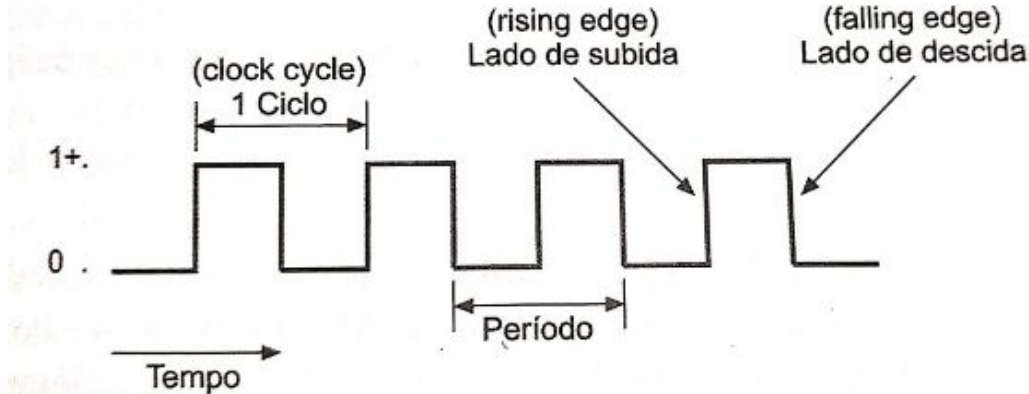
- As microoperações levam, geralmente, um ciclo do sinal de *clock*. Várias microoperações podem ser realizadas conjuntamente;

- As microoperações podem ser geradas a partir de um processo de decodificação com execução direta da instrução (usando um circuito combinacional); ou a partir de um processo de decodificação seguido de interpretação (com um microcódigo);

- No microcódigo são programadas as **microinstruções**, que especificam uma ou mais microoperações.

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

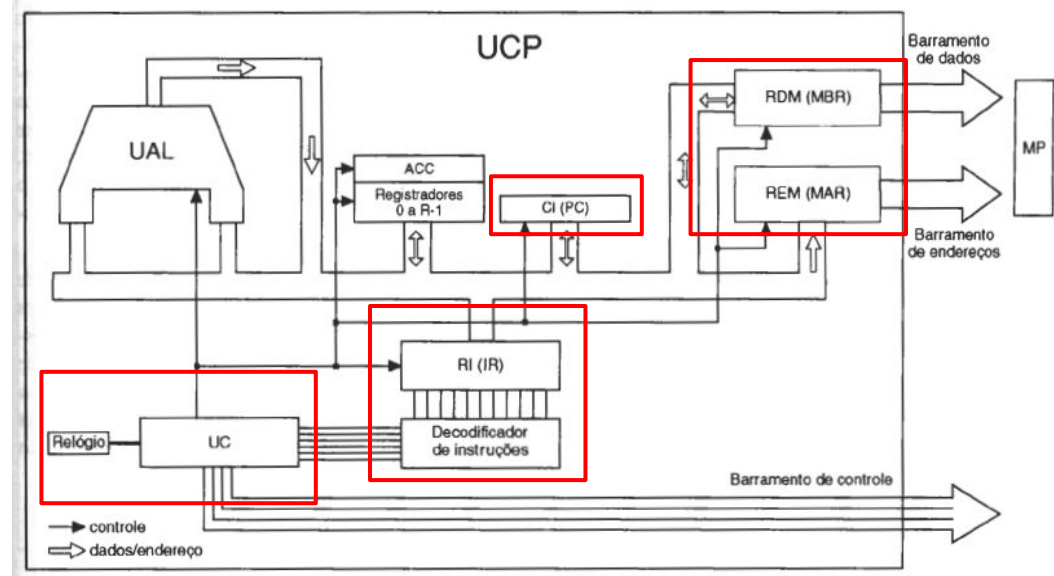
- Em um processador, o que dita o sincronismo das ações é o sinal de *clock*;
- O *clock* é gerado, usualmente, por um **circuito oscilador** onde um de seus elementos é um **cristal de quartzo**, principal componente responsável pelas altas frequências (a partir de MHz) e estabilidade do sinal de *clock*;
- Relembrando: um sinal de *clock* possui uma borda de subida e uma de descida. A distância entre duas bordas de subida ou dias bordas de descida consecutivas nos fornece o **período** do sinal de *clock*.

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Para a unidade de controle gerar os sinais de controle apropriados, além da temporização, ela precisa saber qual instrução deve ser realizada;

- Cada instrução tem um código específico, o **código de operação**, que é guardado no **registrador de instruções (RI)** e, após esse código ser decodificado pelo **decodificador**, um modo de operação específico da unidade de controle é acionada.

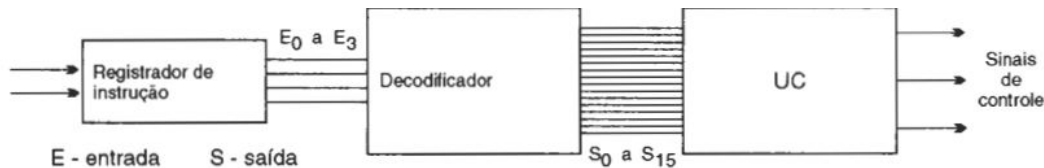
UCP
UAL
ACC
Registradores 0 a R-1
CI (PC)
RDM (MBR)
REM (MAR)
Barramento de dados
Barramento de endereços
MP
RI (IR)
Decodificador de instruções
Relógio
UC
Barramento de controle
controle
dados/endereço

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- A figura ao lado ilustra a relação entre o RI, o decodificador e a UC;

- A tabela verdade mostra que cada código de operação possível no RI ativa somente uma entrada da UC (referente a uma instrução, por exemplo);

- A unidade de controle pode ter outras entradas além das que vem diretamente do decodificador (que podem estar relacionadas aos *flags* do sistema, por exemplo);

- As saídas da UC são os sinais de controle propriamente ditos (que vão definir como o sistema irá operar naquele momento).

(a) Diagrama em bloco da decodificação em uma UCP

E_0	E_1	E_2	E_3	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}	S_{13}	S_{14}	S_{15}
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

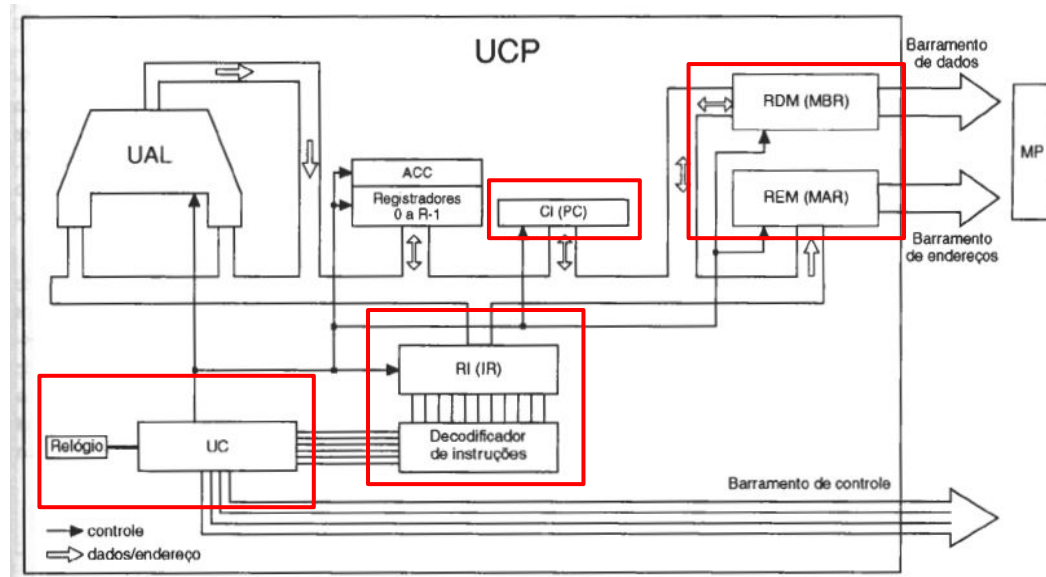
(b) Linhas de Saída

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

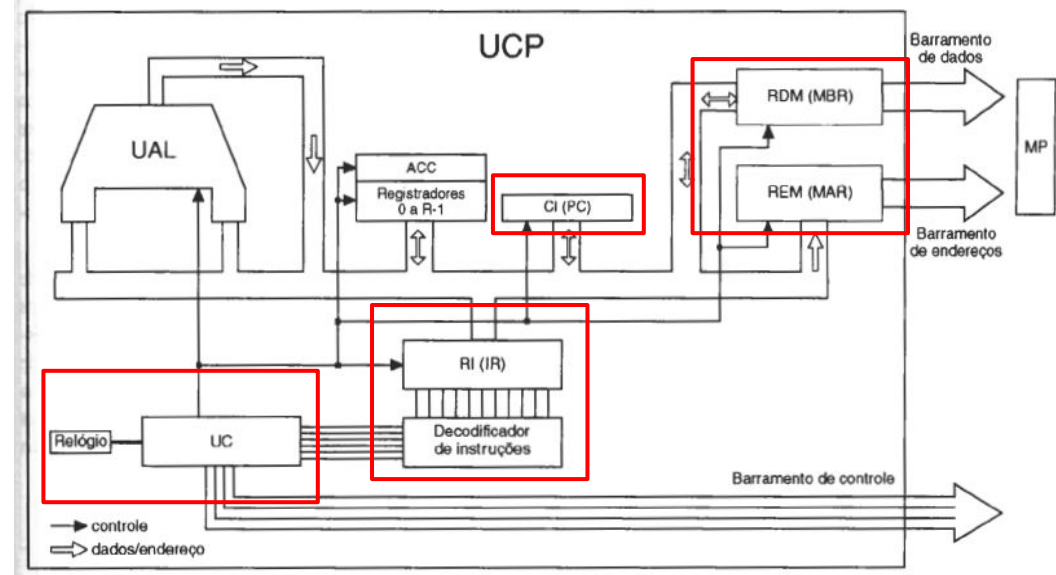
- Como o processador sabe qual instrução acessar? É aí que entra o **Contador de Instruções (CI) ou *Program Counter* (PC)**;

- Quando o sistema é iniciado (ou reiniciado), o registrador CI é inicializado com algum valor, o qual indicará a primeira coisa a ser feita (por exemplo, o sistema pode começar lendo sempre o primeiro endereço da memória de programa);

- Após a instrução ser buscada e salva pelo processador (fim da fase de busca de instrução, ou seja, quando o processador tem a informação do que deve fazer), o registrador CI pode ter o seu valor incrementado,devendo esse processo ocorrer antes da busca da próxima instrução. Após esse incremento, a próxima etapa de busca de instrução irá para o próximo endereço da memória de programa.

  OBS: Algumas instruções alteram o valor do registrador CI para que haja um **salto** no endereço de memória que vai ser lido (veremos isso ao falarmos do nível ISA).

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Por fim, temos mais dois registradores nessa arquitetura genérica: um deles é o **Registrador de Dados de Memória (RDM)** ou ***Memory Buffer Register (MBR)***. O RDM salva os dados que irão entrar ou sair do processador e, portanto, ele possui comunicação direta com o barramento de dados;

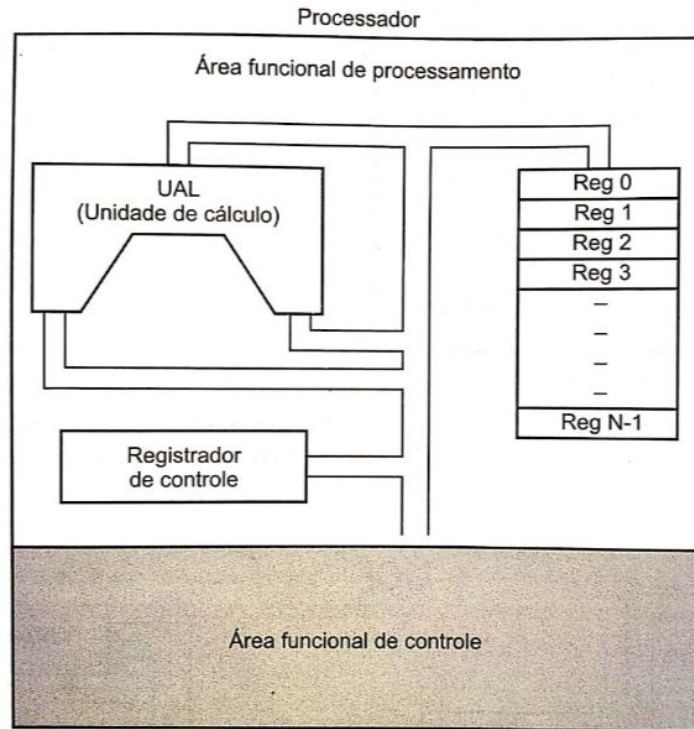
- O outro registrador é o **Registrador de Endereços de Memória (REM)** ou ***Memory Address Register (MAR)***. Ele envia o endereço, especificado pelas instruções do programa, que deve ser utilizado para acessar determinada informação. Esse endereço pode ser relacionado a um dado específico (memória de dados) ou a localização de uma próxima instrução (memória de programa). Nesse último caso, o endereço para acessar a memória de programa vem do registrador CI.

OBS: Os registradores CI e REM devem ter o mesmo tamanho, o qual deve ser, via de regra, o mesmo do barramento de endereços. A partir do tamanho do registrador REM, pode-se saber o número máximo de endereços que podem ser acessados pelo processador em questão. De forma similar, via de regra, o registrador RDM deve ter o mesmo tamanho do barramento de dados.

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

OBS: O **registrador de controle,** apesar de descrito ao falarmos da parte de **processamento dos dados**, na prática, exerce tanto funções de **processamento** quanto de **controle**, de acordo com as *flags* definidas para esse registrador.

OBS2: Esse registrador também é chamado em algumas folhas de dados de **PSW (*Program Status Word*).**

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

- Ao lado, vemos a microarquitetura de um microprocessador AVR (o qual está presente no ATMega328p, microcontrolador da placa Arduino UNO);

- Conseguimos identificar muitos dos blocos genéricos que estudamos: a ULA (ou ALU); os registradores da área de processamento (*General Purpose Registers*); o registrador de controle (*status and control*); o contador de programa (*program counter*); o registrador de instruções (*instruction register*); o decodificador de instruções (*instruction decoder*).

Prof. Alan Sovano

MICROARQUITETURA INTERNA DE UM MICROPROCESSADOR GENÉRICO

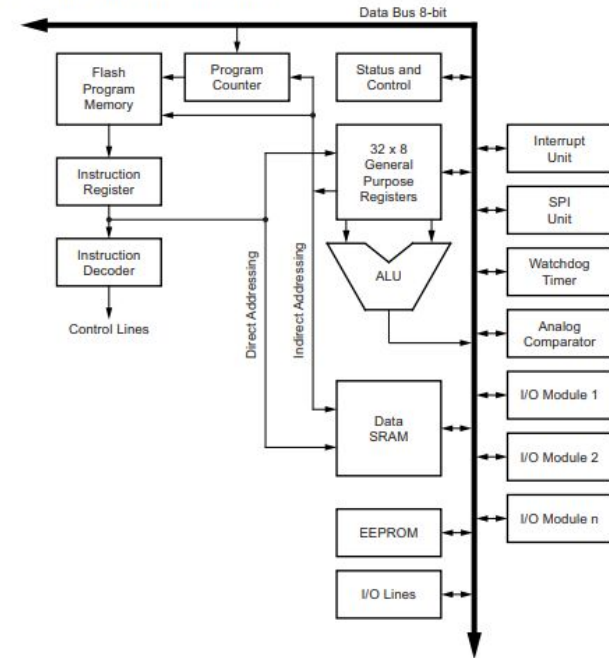
- Vale ressaltar que, no diagrama, o decodificador de instruções é, na verdade, o decodificador de instruções mais a unidade de controle (UC) que vimos em nossa arquitetura genérica. Podemos ver isso pois sua entrada vem do registrador de instruções e sua saída vai para o barramento de controle;

- Perceba também que alguns dos elementos que estudamos não estão ilustrados na figura, mas eles existem! Um exemplo muito claro disso é sinal de *clock*, necessário para todo e qualquer processador, mas omitido no diagrama;

- Isso nos mostra algumas coisas importantes: não há um consenso na melhor forma de mostrar o nível de microarquitetura e, além disso, podem haver nomenclaturas diferentes utilizadas por cada fabricante;

- Por esses motivos, mesmo tendo os conceitos básicos em mente, é sempre interessante estudar a folha de dados do componente que você deseja utilizar!

Figure 6-1. Block Diagram of the AVR Architecture

Prof. Alan Sovano

TIPOS DE DADOS

- Existem tipos específicos de dados com os quais um processador consegue trabalhar. Esse tópico busca mostrar como esses dados são representados e manipulados em um sistema microprocessado;

- Os tipos de dados a serem apresentados são os principais, com muitos processadores possuindo instruções específicas no nível ISA para manipulá-los;

- Vamos começar dividindo os tipos de dados em duas classes: **numéricos** e **não-numéricos**.

TIPOS DE DADOS

- Os dados do tipo **numérico** podem ser representados em uma estrutura chamada de **ponto-fixo** ou em outra chamada de **ponto flutuante**;

- Na estrutura com ponto fixo, o processador entende que a vírgula está **fixada** antes/depois de algum bit específico do registrador. A aritmética com números em ponto fixo ocorre utilizando, geralmente, a **notação em complemento de 2**. Também podem ser utilizadas a **notação sinal-módulo** e a **notação em complemento de 1**;

- Quando falamos em **ponto fixo**, muitas vezes estamos nos referindo aos **números inteiros** – a vírgula está fixada à direita do bit menos significativo (LSB).

TIPOS DE DADOS

- Uma característica importante da representação em complemento de 2 é a seguinte: se tivermos um conjunto de N bits, o número positivo de maior módulo que podemos representar, D_+, e o número negativo de maior módulo que podemos representar, D_-, serão iguais a:

$$D_+ = (2^{N-1}-1)$$

$$D_- = -(2^{N-1})$$

- Nas notações sinal-módulo e complemento de 1, temos que os mesmos números D_+ e D_- são dados por:

$$D_+ = (2^{N-1}-1)$$

$$D_- = -(2^{N-1}-1)$$

- Ou seja, na notação em complemento de 2, somos capazes de representar um valor negativo a mais, sendo uma notação mais eficiente nesse sentido;

- Além disso, é mais fácil construir um circuito que realiza operações aritméticas com números inteiros ao utilizar a notação em complemento de 2 (em comparação com a aritmética baseada na notação em complemento de 1 ou notação sinal-módulo).

TIPOS DE DADOS

- Na estrutura com **ponto flutuante** (onde a vírgula não fica fixada antes/depois de um bit específico), os números são representados em um formato relacionado à **notação científica**, para expressarmos **números reais**. O quadro abaixo representa esse processo.

$$N = \pm M \times B^{\pm E}$$

32 bits

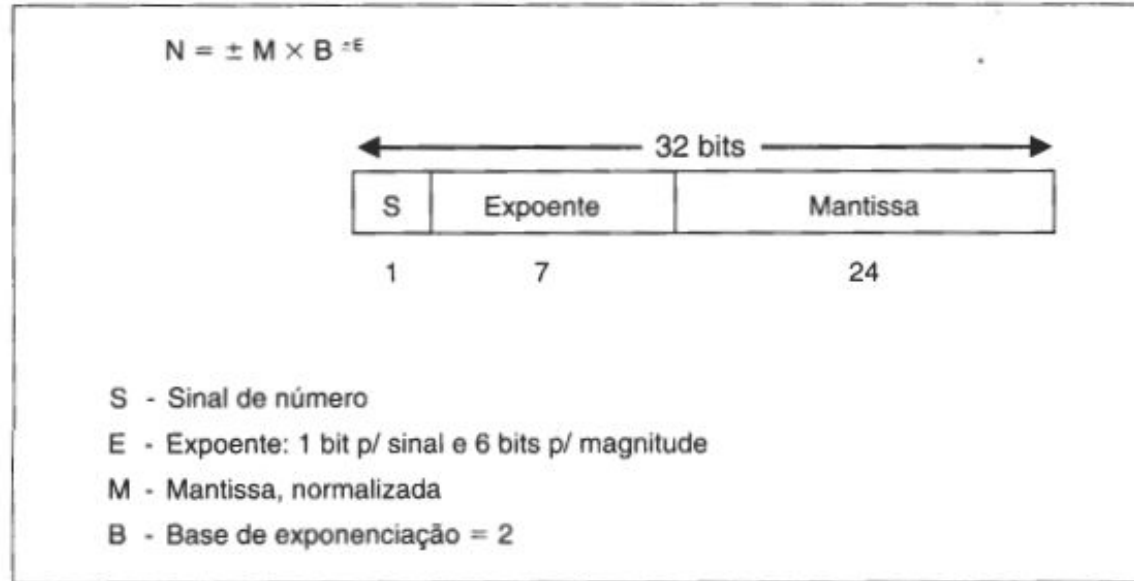
S	Expoente	Mantissa
1	7	24

S - Sinal de número

E - Expoente: 1 bit p/ sinal e 6 bits p/ magnitude

M - Mantissa, normalizada

B - Base de exponenciação = 2

Prof. Alan Sovano

TIPOS DE DADOS

- Os expoentes de um número em ponto flutuante são representados comumente na notação **excesso de N**, a qual elimina a necessidade de um bit de sinal para o expoente;

- Em notação excesso de N, encontra-se o valor C na expressão:

$$C = (2^n/2 - 1) + E$$

- Onde "C" é o número que será armazenado no campo de expoente, "E" é o valor real do expoente e "n" é o número de bits utilizados para armazenar o expoente. O valor $(2^n/2 - 1)$ é o chamado **excesso**;

- A notação em excesso de N faz com que não haja a necessidade de um bit de sinal para o expoente. Além disso, ela facilita a comparação entre os expoentes (um número binário maior representa o expoente maior nesse caso, algo facilmente verificável por um circuito lógico apropriado);

OBS: A mantissa costuma ser representada na notação **sinal-módulo** em operações envolvendo ponto flutuante em um processador.

Prof. Alan Sovano

TIPOS DE DADOS

Exemplo: Uma unidade de ponto flutuante utiliza 8 bits para armazenar expoentes. Para escrever o expoente das potências 2^{10} e 2^{-15}, teríamos que realizar as seguintes operações:

$$C = (2^8/2 - 1) + 10 = 127 + 10 = 137$$

$$C = (2^8/2 - 1) - 15 = 127 - 15 = 112$$

Portanto, o **excesso de 127** do expoente **10** é **137**, enquanto o do expoente **-15** é **112**.

Prof. Alan Sovano

TIPOS DE DADOS

- O padrão mais utilizado para ponto flutuante é o IEEE-754, o qual tem algumas de suas características descritas na tabela abaixo:

Tabela 7.6 Características dos Formatos Básicos de Números em Ponto Flutuante, Representados no Padrão IEEE-754

	Precisão simples	Precisão dupla
Total de bits do número	32 bits	64 bit
Quant. bits para sinal	1 bit	1 bit
Bits para expoente	8	11
Bits para significando	23	52
Cálculo do expoente	Excesso de 127	Excesso de 1023
Faixa de representação em decimal	Aprox. 10^{-38} até 10^{+38}	Aprox. 10^{-308} até 10^{+308}

TIPOS DE DADOS

- No padrão IEEE-754, para escrever a mantissa (na tabela chamada de *significando*), sempre consideramos que ela começa com a parte inteira igual a 1. No campo da mantissa, portanto, sempre colocamos a parte **não-inteira**;

- Escrevemos a parte do expoente, **quando o número é escrito em precisão simples**, sempre utilizando a **notação em excesso de 127**;

- No padrão IEEE-754, algumas notações são reservadas para escrever +∞, -∞ e NaN (*Not a Number* ou "não é um número").

- **OBS:** Um exemplo de operação que retorna NaN é 0/0 (operação sem valor definido).

TIPOS DE DADOS

- Não iremos entrar em detalhes sobre a representação em ponto flutuante e a aritmética em ponto flutuante, mas devemos entender que essa é a representação utilizada na maioria dos processadores que manipulam números que possuem uma parte que não é inteira. Detalhes sobre esse tipo de notação podem ser encontrados na norma IEEE-754 ou em livros que descrevem como trabalhar com ponto flutuante;

- É possível implementar, via software, operações com ponto flutuante em sistemas que possuem somente ULA's que trabalham com números inteiros. Apesar de uma solução viável, não é uma forma eficiente de realizar cálculos com números fracionários;

- Atualmente, os processadores possuem **unidades de ponto flutuante** (UPF ou FPU) implementadas em hardware para trabalhar, especificamente, com operações de números na notação em ponto flutuante.

TIPOS DE DADOS

- Os primeiros processadores não tinham uma FPU embutida na CPU. A FPU era um componente extra, mas que funcionava em conjunto com a CPU, sendo também por isso chamada de **coprocessador**;

- Um **coprocessador** auxilia a função do processador principal. Outro exemplo clássico de coprocessador é a placa de vídeo ou GPU (*Graphical Processing Unit* ou unidade de processamento gráfico);

- Atualmente, muitas CPU's já vem com uma FPU embutida. Em nossos estudos de um microprocessador genérico, não consideramos a FPU como elemento essencial – isso porque um processador é capaz de funcionar sem ela;

- O microcontrolador ATMega328p, por exemplo, não possui uma FPU em sua CPU.

TIPOS DE DADOS

- Computadores podem ter seus desempenhos medidos em FLOPS ou FLOP/s (*floating-point operations per second* ou operações em ponto flutuante por segundo), indicando a velocidade com que um computador realiza cálculos com números em ponto flutuante;

- **OBS:** Outra forma de avaliar a velocidade dos computadores é utilizando a medida de IPS (*Instructions per Second* ou instruções por segundo). Geralmente é utilizado o seu múltiplo, MIPS (ou Mega IPS).

TIPOS DE DADOS

- Para finalizar o nosso estudo sobre FPU's, repare na figura do próximo slide – ela mostra a microarquitetura de um microcontrolador **PIC32MZ EF**, o qual possui um microprocessador **MIPS32 M-class**

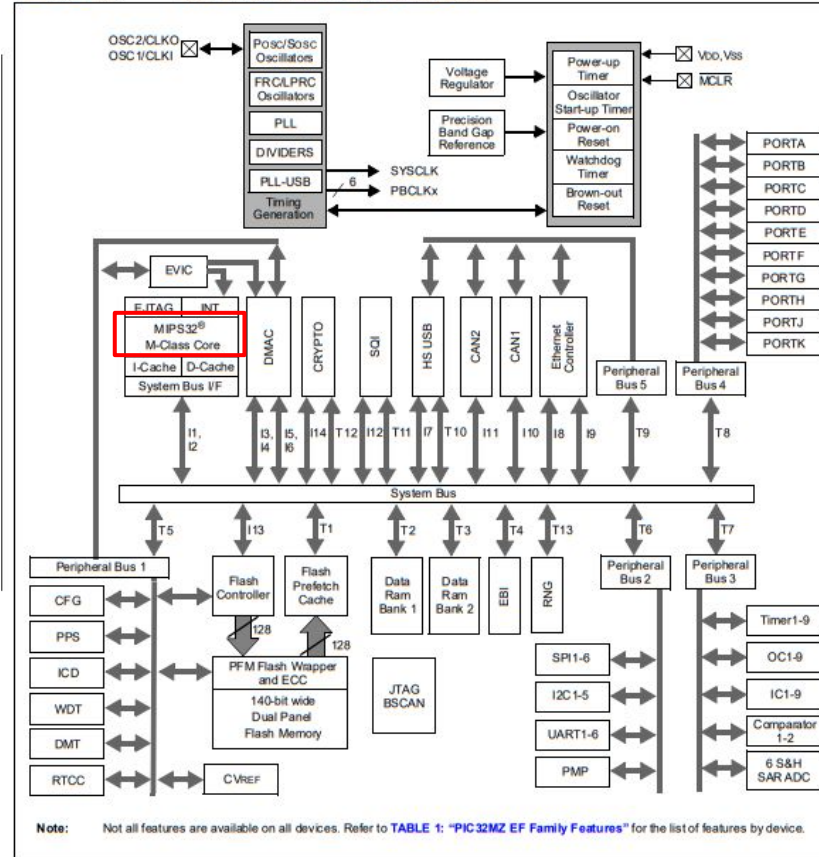
- **OBS:** Aqui, MIPS significa *Microprocessor without Interlocked Pipelined Stages* (Microprocessador sem estágios de *pipeline* intertravados), uma arquitetura de conjunto de instruções (ISA) desenvolvida pela MIPS Technologies (não confunda com a definição anterior de MIPS, que é uma medida de desempenho!). O MIPS32 M-Class é um microprocessador que funciona baseado na arquitetura MIPS.

TIPOS DE DADOS

- Na figura ao lado, temos a microarquitetura de um PIC32MZ EF, o qual possui um microprocessador MIPS32 M-Class;

- O próximo slide mostra a microarquitetura do MIPS32 M-Class.

FIGURE 1-1: PIC32MZ EF FAMILY BLOCK DIAGRAM

Note: Not all features are available on all devices. Refer to **TABLE 1: "PIC32MZ EF Family Features"** for the list of features by device.

TIPOS DE DADOS

- Podemos perceber que, entre várias coisas, o MIPS32 M-Class possui a FPU, como estávamos discutindo!

- Perceba – um microprocessador pode ter muitos elementos em sua microarquitetura, ficando cada vez mais complexo;

- Para utilizar bem um microprocessador e/ou um microcontrolador, deve-se realizar sempre um estudo da folha de dados do componente!

Figure 50-3: M-Class Microprocessor Core Block Diagram

M5150 Microprocessor Core

PBCLK7

Decode (MIPS32®/microMIPS™)

microMIPS™

I-Cache Controller

I-Cache(2)

Execution Unit

ALU/Shift

Atomic/LdSt

DSP ASE(3)

GPR (8 sets)

Enhanced MDU (with DSP ASE(3))

FPU (Single & Double)

MMU (TLB)(1)

BIU

System Bus

D-Cache Controller

D-Cache(2)

System Interface

Interrupt Interface

System Coprocessor

Debug/Profiling

Break Points

iFlowtrace®

Fast Debug Channel

Performance Counters

Sampling

Secure Debug

Power Management

2-wire Debug

EJTAG

Note 1: TLB is available only on devices with the MPU Microprocessor core.

2: Level One (L1) caches are available only on devices with the MPU Microprocessor core.

3: DSP ASE is not available on all devices. Refer to the **"CPU"** chapter in the specific device data sheet to determine availability.

Prof. Alan Sovano

TIPOS DE DADOS

- Além de números em ponto fixo e em ponto flutuante, outra forma de representar dados numéricos é com uma codificação em decimal;

- Pense o seguinte: alguns números decimais, como $3{,}2_{10}$, não podem ser exatamente convertidos para binário – se você tentar realizar essa conversão, chegará no número $(11{,}001100110011\ldots)_2$, uma dízima periódica (binária). Como um computador tem um número limitado de bits, ele não consegue representar **exatamente** o valor $3{,}2_{10}$ dessa forma – ele irá truncar ou arredondar o valor;

- Em algumas aplicações isso não é viável. Truncamentos e arredondamentos introduzem **erros** nos nossos cálculos. Se truncarmos o número $(11{,}001100110011\ldots)_2$ e escrevermos ele como $11{,}0011_2$, ao realizar a conversão de volta para decimal, obtemos o número $3{,}1875_{10}$, ficando evidente que o processo de conversão resultou em um erro, em uma perda de informação;

- Imagine uma calculadora sendo utilizada em um estabelecimento comercial que apresentasse esse tipo de erro: ou o estabelecimento ou o cliente acabariam perdendo dinheiro!

TIPOS DE DADOS

- Por esse motivo, em alguns sistemas (principalmente relacionados a cálculos envolvendo valores monetários), há um dado numérico do tipo **decimal**;

- Na prática, há um processo de codificação – cada dígito decimal é representado por um número binário (o que é diferente de representar o valor diretamente em binário);

- Um exemplo desse processo é a codificação BCD (*Binary Coded Decimal*), a qual representa cada dígito decimal através de uma combinação de 4 bits;

- Portanto, para representar um dado numérico do tipo decimal com dois dígitos seria necessário, por exemplo, um registrador com 8 bits.

TIPOS DE DADOS

- A figura ao lado mostra como a codificação BCD ocorre;
- Realizar operações aritméticas utilizando dados em BCD é algo mais complexo do que realizar operações com números em ponto-fixo (inteiros);
- A aritmética utilizando dados decimais codificados em BCD pode exigir instruções e/ou hardware específico.

OBS: é possível implementar, via software, operações com dados do tipo decimal em sistemas que trabalham unicamente com dados em ponto-fixo (inteiros).

Decimal	Binário — BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

TIPOS DE DADOS

- Além dos dados numéricos, tempos os dados não-numéricos. Temos dois tipos de dados não-numéricos: os **lógicos** e os **caracteres;**

- Os dados lógicos são representados de forma muito simples – um bit que tem seu valor igual a **1** representa uma condição **verdadeira**, enquanto um bit com um valor igual a **0** representa uma condição **falsa**;

- **Exemplo:** se a ULA de um processador faz a operação lógica AND entre o conteúdo de dois registradores de 4 bits, onde o primeiro registrador possui os bits é 1100 e o segundo possui 0101, a saída da ULA será:

	1	**1**	**0**	**0**
AND	**0**	**1**	**0**	**1**
	0	**1**	**0**	**0**

TIPOS DE DADOS

- Temos, por fim, os dados do tipo caractere. Assim como no dados numéricos decimais, os dados do tipo caractere precisam de uma codificação;

- Podemos citar duas codificações comuns: **ASCII** (*American Standard Code for Information Interchange*) e o **UNICODE**;

- O ASCII representa os caracteres em uma codificação que utiliza 7 bits. Há também o ASCII estendido, o qual utiliza 8 bits (incluindo mais caracteres);

- A codificação ASCII possui, majoritariamente, caracteres associados ao alfabeto latino. Para outros tipos de caracteres (que incluem o alfabeto grego, por exemplo), é utilizada a codificação UNICODE, a qual pode utilizar 8, 16 ou até 32 bits;

- Os processadores costumam possuir instruções específicas para manipular cadeias de caracteres (*strings*).

OBS: Nas *strings*, cada caractere é geralmente representado por 1 byte (8 bits). Para indicar o término da *string*, há um caractere nulo em seu fim (onde o valor de todos os bits do byte é igual a zero), geralmente representado por “\0”.

TIPOS DE DADOS

most significant nibble / *least significant nibble*

	0_	1_	2_	3_	4_	5_	6_	7_
_0	NUL	DLE	SP	0	@	P	'	p
_1	SOH	DC1	!	1	A	Q	a	q
_2	STX	DC2	"	2	B	R	b	r
_3	ETX	DC3	#	3	C	S	c	s
_4	EOT	DC4	$	4	D	T	d	t
_5	ENQ	NAK	%	5	E	U	e	u
_6	ACK	SYN	&	6	F	V	f	v
_7	BEL	ETB	'	7	G	W	g	w
_8	BS	CAN	(	8	H	X	h	x
_9	HT	EM	)	9	I	Y	i	y
_A	LF	SUB	*	:	J	Z	j	z
_B	VT	ESC	+	;	K	[	K	}
_C	FF	FS	,	<	L	\	l	|
_D	CR	GS	-	=	M	]	m	}
_E	SO	RS	.	>	N	^	n	~
_F	SI	US	/	?	O	_	o	DEL

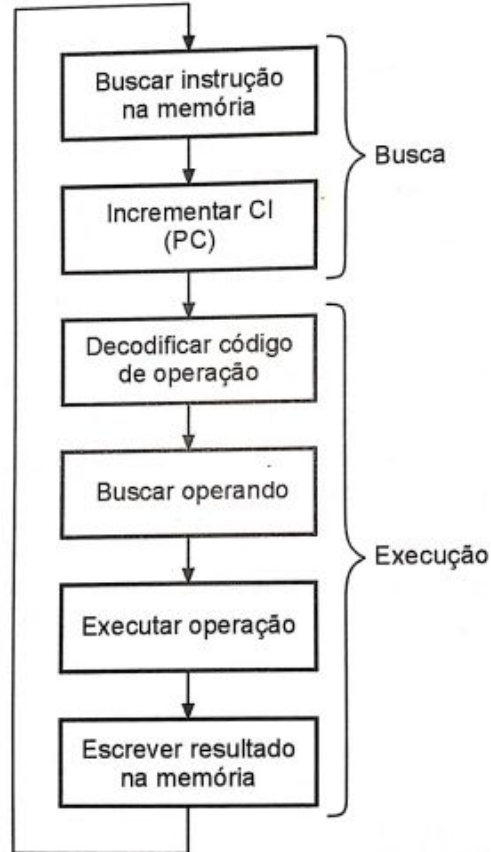
- A tabela ao lado mostra como a codificação em ASCII ocorre. Cada caractere está codificado com dois dígitos hexadecimais, representando 8 bits;
- Nessa codificação do ASCII, o bit mais significativo é sempre 0 (os caracteres são representados, de fato, por 7 bits, com 1 bit extra para completar o tamanho total de 1 byte).

PIPELINING

- Ao estudar o ciclo de busca-decodificação-execução, vimos que o processador, a princípio, funciona de forma sequencial;

- Acontece que existem etapas desse ciclo que podem ser feitas ao mesmo tempo para instruções diferentes. Por exemplo, enquanto uma instrução está na fase de decodificação, a próxima já poderia entrar na fase de busca de instrução;

- Esse procedimento acontece nos processadores atuais, os quais funcionam como uma **linha de montagem** (um ***pipeline***), sofrendo um processo de **canalização** ou ***pipelining*** de instruções.

PIPELINING

- Considere um ciclo de instrução dado pelo fluxograma ao lado, o qual será utilizado como base para discutir o *pipelining*.

PIPELINING

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Instrução 1	BI	CI	DE	BO	EO	RE											
Instrução 2		BI	CI	DE	BO	EO	RE										
Instrução 3			BI	CI	DE	BO	EO	RE									
Instrução 4				BI	CI	DE	BO	EO	RE								
Instrução 5					BI	CI	DE	BO	EO	RE							
Instrução 6						BI	CI	DE	BO	EO	RE						
Instrução 7							BI	CI	DE	BO	EO	RE					
Instrução 8								BI	CI	DE	BO	EO	RE				
Instrução 9									BI	CI	DE	BO	EO	RE			
Instrução 10										BI	CI	DE	BO	EO	RE		

Tempo →

BI - Busca interna
CI - Calcula CI
DE - Decodificador C.Op.
BO - Busca operando
EO - Executa operação
RE - Escreve resultado

- Se cada etapa do ciclo de instrução for executada por uma parte diferente do sistema, várias instruções podem ser executadas ao mesmo tempo;

- A primeira instrução levará, em nosso exemplo, 6 ciclos de relógio para ser executada (considerando que cada etapa leva um ciclo de *clock*);

- Após a execução da primeira instrução, com o *pipelining*, a cada próximo ciclo de *clock*, uma nova instrução será finalizada, aumentando a eficiência do microprocessador;

- Esse *pipeline*, idealmente, realiza todas as 6 etapas do ciclo de instrução ao mesmo tempo, com cada etapa sendo referente a uma instrução diferente!

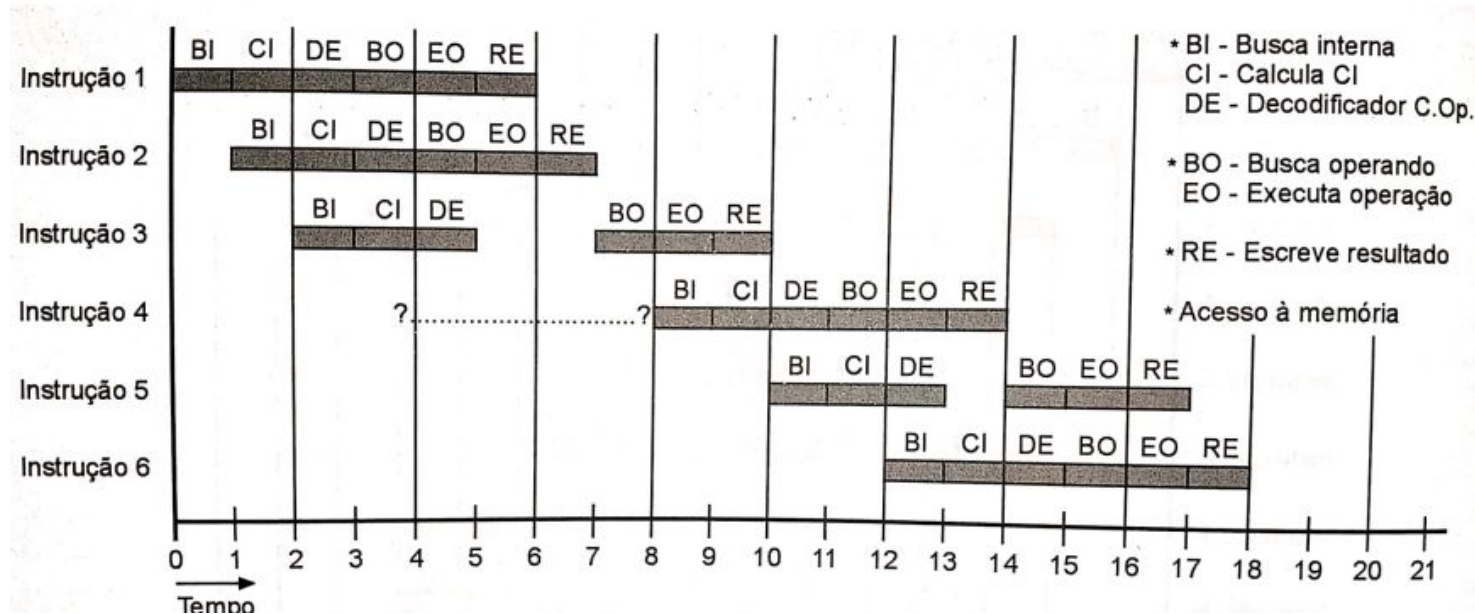
PIPELINING

- Na prática, não temos como realizar todas as etapas ao mesmo tempo – algumas necessitam das mesmas partes do sistema.

Instrução 1: BI CI DE BO EO RE
Instrução 2: BI CI DE BO EO RE
Instrução 3: BI CI DE BO EO RE
Instrução 4: ? ? BI CI DE BO EO RE
Instrução 5: BI CI DE BO EO RE
Instrução 6: BI CI DE BO EO RE

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
Tempo

* BI - Busca interna
CI - Calcula CI
DE - Decodificador C.Op.
* BO - Busca operando
EO - Executa operação
* RE - Escreve resultado
* Acesso à memória

Prof. Alan Sovano

PIPELINING

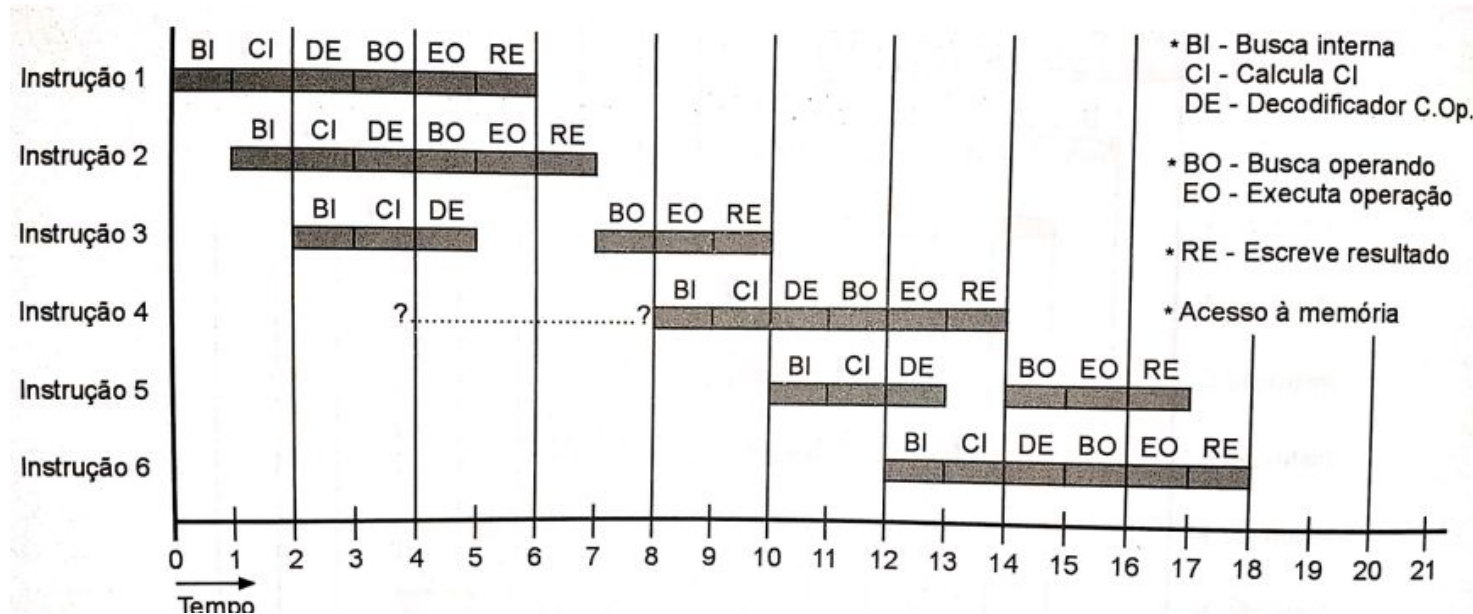
- Vamos considerar, agora, que as etapas BI, BO e RE precisam acessar o barramento de dados para se comunicar com a memória principal (leitura/escrita de dados na memória principal). Nesse caso (mais realista), haverá um conflito de acesso à memória por estágios diferentes.

Instrução 1: BI CI DE BO EO RE
Instrução 2: BI CI DE BO EO RE
Instrução 3: BI CI DE ... BO EO RE
Instrução 4: ?...............? BI CI DE BO EO RE
Instrução 5: BI CI DE ... BO EO RE
Instrução 6: BI CI DE BO EO RE

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

Tempo

- * BI - Busca interna
  CI - Calcula CI
  DE - Decodificador C.Op.
- * BO - Busca operando
  EO - Executa operação
- * RE - Escreve resultado
- * Acesso à memória

Prof. Alan Sovano

PIPELINING

- Dessa forma, a eficiência do *pipeline* é limitada por diversos fatores práticos.

Instrução 1: BI CI DE BO EO RE
Instrução 2: BI CI DE BO EO RE
Instrução 3: BI CI DE ... BO EO RE
Instrução 4: ?.....? BI CI DE BO EO RE
Instrução 5: BI CI DE ... BO EO RE
Instrução 6: BI CI DE BO EO RE

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

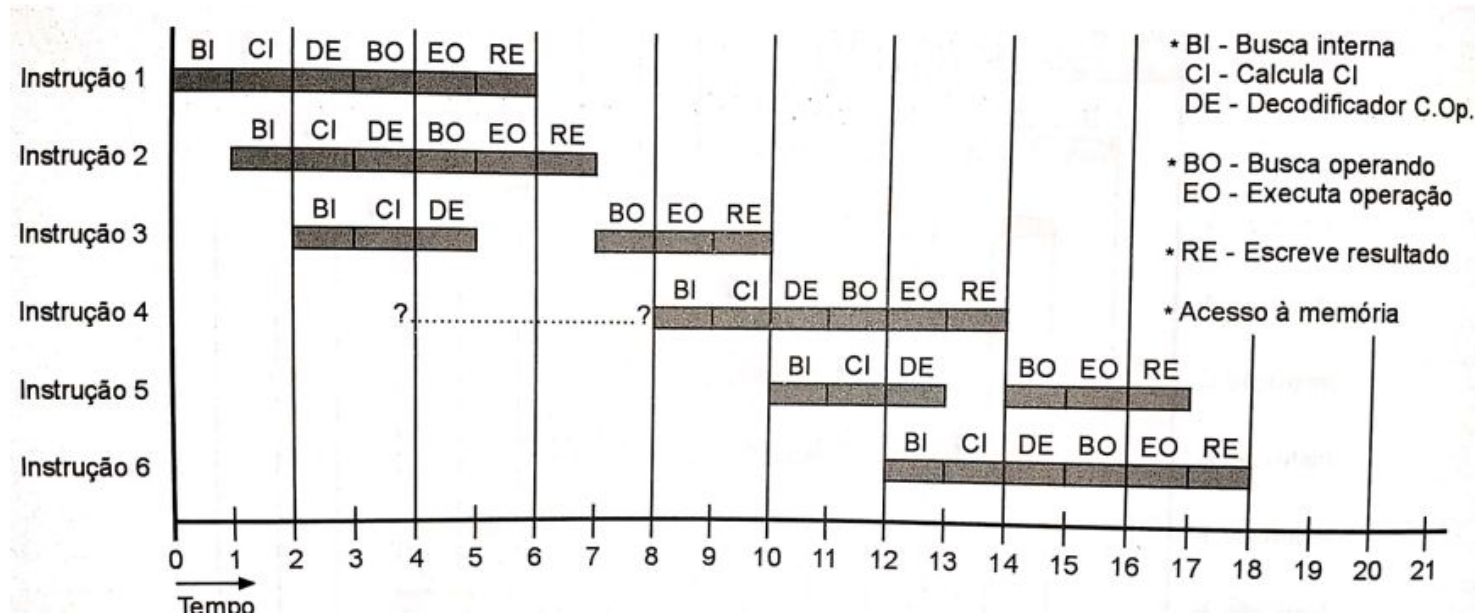
Tempo

* BI - Busca interna
CI - Calcula CI
DE - Decodificador C.Op.

* BO - Busca operando
EO - Executa operação

* RE - Escreve resultado

* Acesso à memória

Prof. Alan Sovano

PIPELINING

- Em nosso exemplo, consideramos que a busca de instruções e de dados ocorre pelo mesmo barramento (processador operando em um computador com arquitetura de Von-Neumann);

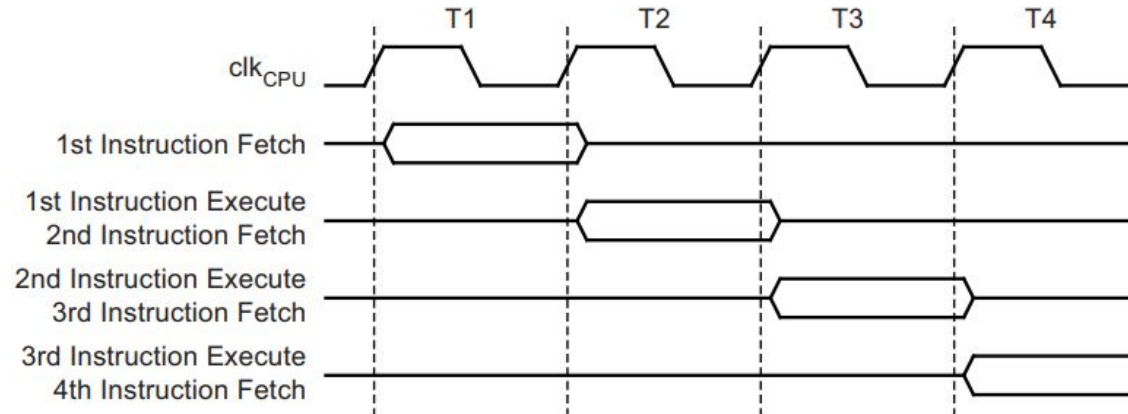
- Se houvesse uma separação da memória de programa (instruções) da de dados (arquitetura Harvard), haveria a possibilidade de obter um *pipelining* mais eficiente;

- Logo, podemos concluir que a arquitetura Harvard **é uma solução melhor** para a implementação de um *pipeline* em comparação com a de Von-Neumann;

- **OBS:** Na prática, outros fatores também limitam o *pipelining*, como o tempo de duração de cada etapa (que não é necessariamente igual) ou a utilização de instruções de **salto** ou **desvio** (que altera a ordem de execução de instruções). Há também a possibilidade da próxima instrução precisar de dados da instrução anterior, havendo um atraso na execução dessa próxima instrução. Todos esses problemas devem ser considerados ao implementar um *pipeline*.

PIPELINING

- No *datasheet* do ATMega328p, é descrito que seu microprocessador tem um *pipeline* simples de dois estágios: enquanto há a busca de uma instrução, há a execução de outra. Lembre-se: o ATMega328p é um microcontrolador que possui arquitetura Harvard!

Prof. Alan Sovano

PIPELINING

- Quando falamos da implantação de um *pipelining* no processador, estamos tentando fazer com que o processador seja capaz de fazer mais coisas ao mesmo tempo. É uma tentativa de processar mais informação ao mesmo tempo, ou seja, em **paralelo**;

- Os problemas que podem ocorrer em um *pipeline* também são chamados de ***hazards*** (**perigos**). Existem três tipos de *hazards* possíveis: os ***hazards* estruturais**, ***hazards* de dados** e ***hazards* de controle**.

PIPELINING

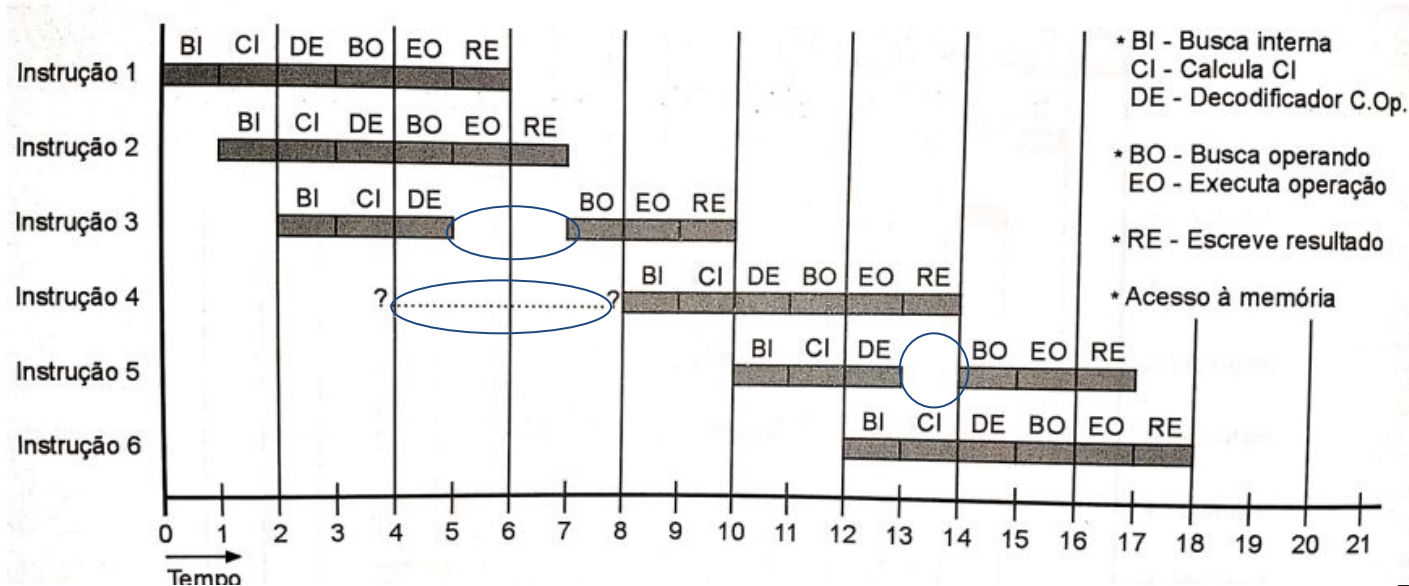
- *Hazards* estruturais ocorrem quando estágios diferentes do *pipeline* precisam de um mesmo recurso (exemplo: duas instruções que precisam acessar a memória ao mesmo tempo);

- *Hazards* de dados ocorrem quando há uma dependência de dados entre instruções. (exemplo: quando uma instrução de soma gera uma valor qualquer, e a próxima instrução depende desse valor – nesse caso, a próxima instrução só poderá ser executada quando a primeira for finalizada);

- *Hazards* de controle ocorrem quando instruções entram no *pipeline* sem necessidade, e acabam sendo descartadas (exemplo: quando um instrução de desvio aparece no programa, mudando a sequência de instruções que devem ser executadas).

PIPELINING

- Existem várias técnicas para se lidar com *hazards* em pipelines. Uma delas é a inserção de bolhas (*bubbles*), que são momentos ociosos dentro do *pipeline*. Retomando o nosso exemplo anterior do *pipeline* com conflitos de acesso à memória (*hazard* estrutural), podemos ver as bolhas (em azul) inseridas para evitar problemas durante o processamento.

Prof. Alan Sovano

PROCESSADORES COM MÚLTIPLOS *CORES* E MÚLTIPLOS *THREADS*

- Os gargalos do *pipelining* levaram ao desenvolvimento de outras técnicas complementares de **paralelização**;

- Uma delas é a utilização de múltiplos núcleos (***cores***) em um único circuito integrado. Cada núcleo funciona como um processador independente, com um único circuito integrado sendo capaz de realizar várias ***threads*** ao mesmo tempo, associados a um mesmo **processo**.

PROCESSADORES COM MÚLTIPLOS *CORES* E MÚLTIPLOS *THREADS*

- O conceito de processo está associado ao estudo de sistemas operacionais. Um processo pode ser definido como a instância de um programa em execução;

- Um programa, ao ser executado, tem, de forma geral, recursos alocados (como memória, contador de programa, registradores para processamento, entre outros), um estado (aguardando execução, em execução de fato ou bloqueado) e um identificador (PID ou identificador de processo). Esse conjunto de fatores necessários para caracterizar um programa que foi colocado em execução é o que define um processo;

- Quando você clica no ícone para abrir um editor de texto, por exemplo, você está iniciando um processo – você começou a execução de um programa. Se você clicar de novo no ícone do editor e “abrir o programa de novo”, você está iniciando um novo processo de um mesmo programa.

PROCESSADORES COM MÚLTIPLOS *CORES* E MÚLTIPLOS *THREADS*

- Esse mesmo editor de texto pode ter seu processo subdividido em etapas que podem ocorrer ao mesmo tempo: o salvamento dos arquivos automaticamente, a correção da ortografia e recebimento de caracteres de entrada;

- Cada uma desses *caminhos de execução* é independente um do outro, podendo todos serem processados ao mesmo tempo, mas estão todos relacionados ao mesmo **processo** (compartilham recursos alocados para o processo). Essas subdivisões de etapas são as ***threads***;

- Cada *thread* pode ser executada por um núcleo de processamento, aumentando a capacidade de processamento em paralelo do sistema;

- Todo processo deve ter ao menos uma *thread*, mas pode ter várias.

PROCESSADORES COM MÚLTIPLOS *CORES* E MÚLTIPLOS *THREADS*

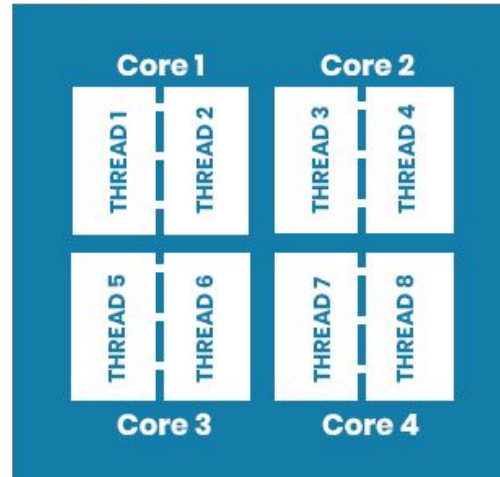
- Sistemas com múltiplos núcleos suportam *multithreading*, onde cada núcleo é capaz de trabalhar com uma *thread*, aumentando o paralelismo da execução de um programa e, consequentemente, fazendo com que o sistema faça mais coisas ao mesmo tempo;

- Existem processadores onde um núcleo de processamento consegue executar mais de uma *thread*, criando a ideia de um "núcleo virtual" de processamento;

- Quando um processador tem 8 *cores* e 16 *threads*, por exemplo, significa que o sistema tem 8 núcleos físicos de processamento, com cada um suportando, a princípio, 2 *threads* por vez;

- Essa é a base do funcionamento de técnicas como o *hyperthreading* da Intel.

PROCESSADORES COM MÚLTIPLOS *CORES* E MÚLTIPLOS *THREADS*

- Quando um núcleo físico executa duas *threads* de uma vez, na prática, essas *threads* não são executadas perfeitamente em paralelo – uma acaba, em algum momento, precisando esperar a outra ser executada. Quando temos vários núcleos físicos, a execução (comparando um núcleo com outro) pode ser perfeitamente paralela.

OBRIGADO PELA ATENÇÃO!

Dúvidas ou sugestões?

sovano@ufpa.br

Prof. Alan Sovano